Gross Store

Welcome to Gross Store on Exercism's Go Track. If you need help running the tests or submitting your code, check out `HELP.md`. If you get stuck on the exercise, check out `HINTS.md`, but try and solve it without using those first :)

Introduction

In Go, `map` is a built-in data type that maps keys to values. In other programming languages, you might be familiar with the concept of `map` as a dictionary, hash table, key/value store or an associative array.

Syntactically, `map` looks like this:

```
map[KeyType]ElementType
```

It is also important to know that each key is unique, meaning that assigning the same key twice will overwrite the value of the corresponding key.

To create a map, you can do:

```
  // With map literal
  foo := map[string]int{}
```

or

```
  // or with make function
  foo := make(map[string]int)
```

Here are some operations that you can do with a map

```
  // Add a value in a map with the `=` operator:
  foo["bar"] = 42
  // Here we update the element of `bar`
  foo["bar"] = 73
  // To retrieve a map value, you can use
```

```
    baz := foo["bar"]
    // To delete an item from a map, you can use
    delete(foo, "bar")
```

If you try to retrieve the value for a key which does not exist in the map, it will return the zero value of the value type. This can confuse you, especially if the default value of your `ElementType` (for example, 0 for an int), is a valid value. To check whether a key exists in your map, you can use

```
    value, exists := foo["baz"]
    // If the key "baz" does not exist,
    // value: 0; exists: false
```

Instructions

A friend of yours has an old wholesale store called **Gross Store**. The name comes from the quantity of the item that the store sell: it's all in gross unit. Your friend asked you to implement a point of sale (POS) system for his store. **First, you want to build a prototype for it. In your prototype, your system will only record the quantity.** Your friend gave you a list of measurements to help you:

Unit	Score
quarter_of_a_dozen	3
half_of_a_dozen	6
dozen	12
small_gross	120
gross	144
great_gross	1728

1. Store the unit of measurement in your program

In order to use the measurement, you need to store the measurement in your program.

```
units := Units()
fmt.Println(units)
// Output: map[...] with entries like ("dozen": 12)
```

2. Create a new customer bill

You need to implement a function that create a new (empty) bill for the customer.

```
bill := NewBill()
fmt.Println(bill)
// Output: map[]
```

3. Add an item to the customer bill

To implement this, you'll need to:

- Return `false` if the given `unit` is not in the `units` map.
- Otherwise add the item to the customer `bill` , indexed by the item name, then return `true` .
- If the item is already present in the bill, increase its quantity by the amount that belongs to the provided `unit` .

```
bill := NewBill()
units := Units()
ok := AddItem(bill, units, "carrot", "dozen")
fmt.Println(ok)
// Output: true (since dozen is a valid unit)
```

> Note that the returned value is type `bool` .

4. Remove an item from the customer bill

To implement this, you'll need to:

- Return `false` if the given item is **not** in the bill

- Return `false` if the given `unit` is not in the `units` map.
- Return `false` if the new quantity would be less than 0.
- If the new quantity is 0, completely remove the item from the `bill` then return `true` .
- Otherwise, reduce the quantity of the item and return `true` .

```go
bill := NewBill()
units := Units()
ok := RemoveItem(bill, units, "carrot", "dozen")
fmt.Println(ok)
// Output: false (because there are no carrots in the bill)
```

> Note that the returned value is type `bool` .

5. Return the quantity of a specific item that is in the customer bill

To implement this, you'll need to:

- Return `0` and `false` if the `item` is not in the bill.
- Otherwise, return the quantity of the item in the `bill` and `true` .

```go
bill := map[string]int{"carrot": 12, "grapes": 3}
qty, ok := GetItem(bill, "carrot")
fmt.Println(qty)
// Output: 12
fmt.Println(ok)
// Output: true
```

> Note that the returned value are types `int` and `bool` .

Source

Created by

- @chocopowwwa

Contributed to by

- @MiroslavGatsanoga